Journal of Theoretical and Applied Information Technology
15th May 2024. Vol.102. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DESIGN AND IMPLEMENTATION OF A CONTAINER ORCHESTRATION SYSTEM FOR DISTRIBUTED REINFORCEMENT LEARNING DATA ANALYSIS

SEOK JAE MOON, SEO YEON GU†

Department of Artificial Intelligence, Institute of Information Technology, Kwangwoon University, Korea
†Department of Computer Science, Kwangwoon University, Seoul, Korea. (Corresponding author)

E-mail: msj80386@kw.ac.kr, ksy06136@gmail.com

ABSTRAC T

Recently, reinforcement learning has shown excellent performance in solving complex data analysis problems in the real world, and many companies are actively introducing it. However, as the diversity and complexity of corporate business processes continues to increase, existing reinforcement learning approaches have limitations in solving complex problems. To cope with this, more sophisticated algorithms have been developed, but these algorithms require high computational resources. As a result, many enterprises seek to obtain the computing resources they need by leveraging distributed environments such as the cloud. However, as the diversity and complexity of enterprise business processes increase, the workload that cloud service providers must manage becomes more complex. Therefore, container orchestration mechanisms are becoming more complex and resource utilization is becoming more difficult. Therefore, in this study, we propose a container orchestration system for distributed reinforcement learning data analysis. This proposed system consists of User Interface, Task Processing Layer, and Infrastructure. In addition, through performance comparison experiments with existing centralized processing methods, that the proposed system is suitable for data analysis in a distributed environment.

Keywords: *Container Orchestration, Distributed Reinforcement Learning, Data Analysis*

1. INTRODUCTION

Recently, numerous companies and data scientists have been exploring various machine learning (ML) algorithms, including supervised learning, unsupervised learning, and reinforcement learning, to glean valuable insights from big data [1]. Reinforcement learning, in particular, has demonstrated its efficacy in addressing real-world challenges such as stock trading, robot control, and natural language processing, leading to its widespread adoption by many companies [2]. However, as the diversity and complexity of corporate business processes increase, existing reinforcement learning algorithms encounter limitations in tackling intricate problems [3]. To address this, more sophisticated algorithms like DQN, A3C, and IMPALA [4] have been investigated. Yet, executing these complex algorithms demands substantial computational resources, necessitating high-performance computing infrastructure [5]. While large enterprises develop and utilize independent infrastructure for data analysis, smaller companies find it economically and temporally challenging to build such infrastructure, leading many to adopt cloud environments instead [6].

Many data centers and corporate infrastructures are now hosted in the cloud, where computing resources are provisioned as cluster computing over the network [7]. Cloud service providers leverage virtualization technologies such as virtual machines (VMs) and containers in distributed infrastructures to facilitate automated application deployment [8]. Containers, which isolate processes at the operating system (OS) level, offer a lighter and more space-efficient alternative to VMs, which virtualize resources at the hardware level [20]. Additionally, containers streamline data analysis services by encapsulating datasets and analysis software into portable containers, offering convenience to developers, especially in small businesses [9]. However, as the complexity of corporate business

processes continues to grow, managing the workload handled by cloud service providers in terms of both quantity and quality becomes challenging [8, 10], significantly complicating container orchestration mechanisms and hindering small businesses' ability to manage the data analysis process effectively.

Research on policy optimization for efficient workload management in container orchestration is ongoing, with many studies emphasizing the need for behavioral modeling and prediction of multidimensional performance indicators using RL [2, 8]. The integration of RL enables efficient orchestration, including maximizing application processing and ensuring fair resource allocation. However, existing research lacks a systematic approach to building a framework fully optimized for RL, highlighting the need for further investigation in this area [8].

In this study, we propose and design a container orchestration system tailored for distributed reinforcement learning data analysis. The proposed system comprises three layers: User Interface, Task Processing Layer, and Infrastructure. When a user initiates a process, the Distributed RL Engine extracts the RL component and converts it into an application, which is then containerized. Upon deployment and execution of the container in a cluster, the execution results are relayed to the Distributed RL Engine, which aggregates multiple results and visualizes the data for user review. The Container Orchestration module is responsible for resource monitoring, task scheduling, cluster management, low latency, and distributed data storage. The low-latency overcoming module utilizes the Safe Proper Time (SPT) method to mitigate the impact of network latency [11].

This paper is organized into five chapters. Chapter 2 discusses related research, Chapter 3 provides an overview of the proposed system and its components, and Chapter 4 conducts experiments to analyze the system's performance. Finally, Chapter 5 presents conclusions and outlines future research directions.

2. RELATED WORK

2.1 Distributed Reinforcement Learning

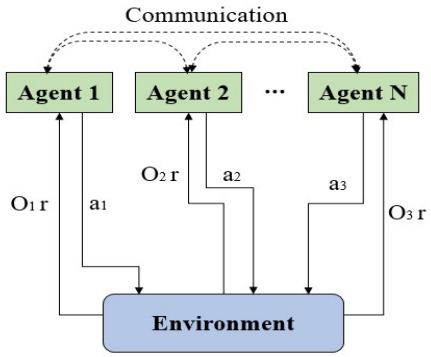


Figure 1: Distributed reinforcement learning approach

Distributed reinforcement learning involves multiple agents learning their own action policies for individual or shared goals while interacting with a common environment. These agents may learn policies for competitive or collaborative actions depending on their objectives. Unlike single-agent reinforcement learning, distributed reinforcement learning considers interactions not only between the agent and its environment but also between multiple agents [12]. It treats a single problem as multiple local problems based on its distributed nature. In distributed reinforcement learning, the experience of performing a single action can be viewed as multiple vectors, allowing for the acquisition and utilization of large amounts of learning data from a single sampling, thereby enhancing learning efficiency. Furthermore, various action policies learned by individual agents can be shared among agents, facilitating the generalization of action policies [12]. The advantage of distributed reinforcement learning lies in its ability to enable large-scale learning, thereby enhancing the performance of state-of-the-art reinforcement learning algorithms such as A3C and IMPALA [4].

2.2 Cloud Computing and Orchestration

High Throughput Computing (HTC) is a computing paradigm aimed at enhancing throughput by maximizing the utilization of idle computer resources [7]. HTC encompasses both cluster computing and cloud computing, with orchestration and virtualization serving as core technologies in each computing paradigm. Many data centers are structured as clouds utilizing virtualization, and computing resources provided through the cloud can be leveraged for cluster computing purposes [7].

Journal of Theoretical and Applied Information Technology
15th May 2024. Vol.102. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

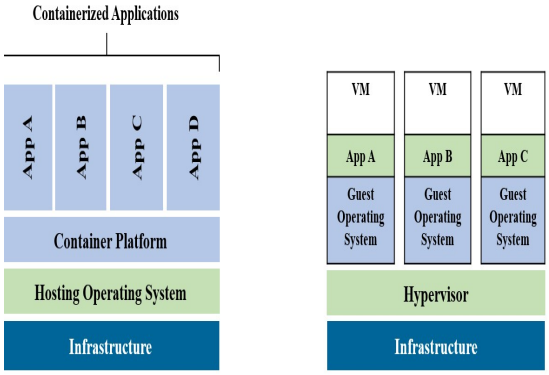


Figure 2: Container and VM configuration

Virtualization involves the abstraction of computer resources, with containers and virtual machines (VMs) serving as representative technologies. Containers virtualize resources at the operating system (OS) level, sharing the OS kernel with the host and offering greater resource efficiency and environment consistency compared to VMs [8]. The advantages of containers have led to a shift in distributed system infrastructure from being VM-centric to container-centric [8]. Consequently, there is an increasing demand for container orchestration research to automate the deployment, maintenance, management, and autoscaling of containerized applications. Particularly, there is active research on integrating machine learning (ML) for efficient container orchestration, with a focus on orchestrating business processes and managing workload units effectively [13].

3. CONTAINER ORCHESTRATION SYSTEM FOR DISTRIBUTED REINFORCEMENT LEARNING

3.1 The Overview of the Proposed System

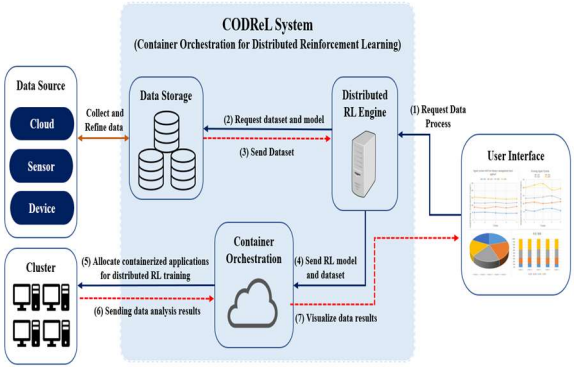


Figure 3: Overview of the proposed system

The overview of the system proposed in this paper is depicted in Figure 3. When a user requests a data analysis process, (1) the Distributed RL Engine receives the process execution request along with the dataset and model from the data storage (2, 3). Subsequently, the container orchestration sends the received dataset and model for execution (4). Container orchestration containerizes the received resources and distributes them to each cluster to ensure the requested process runs smoothly (5). Upon completion of the process, the results are returned (6), and the user can review the desired outcomes (7).

3.2 Components of the proposed system

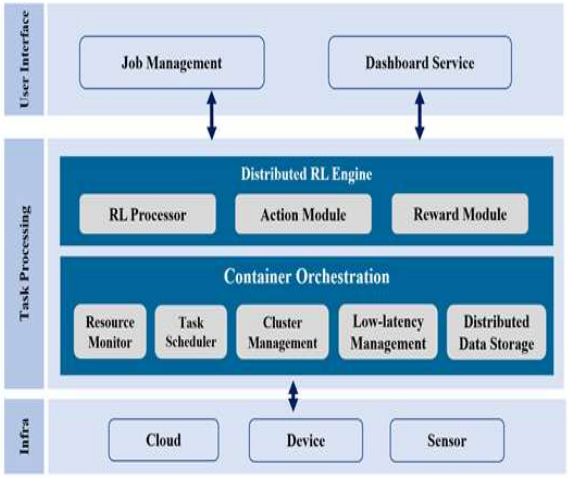


Figure 4: Components of the proposed system

Figure 4 illustrates the configuration of the proposed system, which comprises three layers: the User Interface, Task Processing Layer, and Infrastructure.

User Interface serves as the interface between the user and the Task Processing Layer. It encompasses modules such as Job Management and Dashboard Service.

The Task Processing Layer forms the core of the system, responsible for processing user request processes. It consists of two main components: Container Orchestration and Distributed RL Engine.

The Container Orchestration component includes modules like Resource Monitor, Task Scheduler, Cluster Management, Low-latency Management, and Distributed Data Storage.

The Infrastructure layer serves as the data collection source and encompasses elements such as cloud services, devices, sensors, and more.

3.2 User Interface

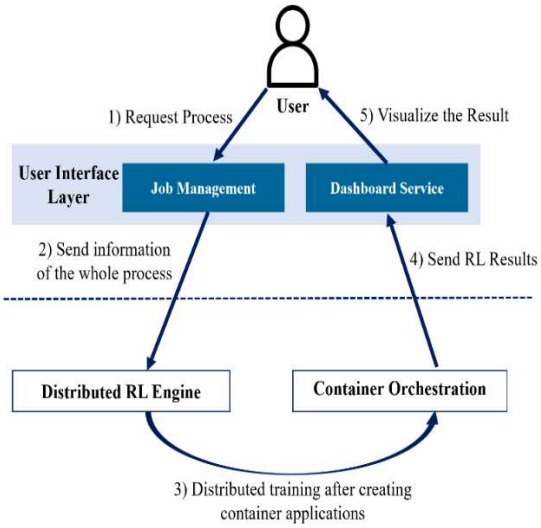


Figure 5: User Interface workflow

User Interface layer specifies process requirements and presents data visualization tools to facilitate informed decision-making by users. It comprises two modules: Job Management and Dashboard Service.

Job Management module serves as an interface between the user and the Distributed RL Engine, transmitting user-defined process information to the Distributed RL Engine for execution.

Meanwhile, Dashboard Service module furnishes a dashboard environment for monitoring resource utilization and execution outcomes, including CPU and network metrics for each cluster. This capability enables the system to monitor inadvertent resource consumption, identify rapid system alterations, and promptly respond to uphold operational consistency within the system.

3.3 Task Processing Layer

Distributed RL Engine generates multiple applications to enable multiple clusters to process user-requested tasks, while Container Orchestration oversees the smooth operation of each cluster.

Comprising the RL Processor, Reward Module, and Action Module, Distributed RL Engine carries out various functions. RL Processor identifies processes, extracts components requiring query and RL analysis, and analyzes these components to determine an appropriate RL algorithm. It then solicits the Distributed Data Storage for the RL model and dataset corresponding to the selected algorithm. Subsequently, it partitions the entire dataset into N subsets and generates N applications containing these subsets. Upon application creation, the Distributed RL Engine dispatches the applications to the Container Orchestration.

Clusters that deploy application containers from Container Orchestration execute the applications and relay the execution outcomes in the form of actions to Distributed RL Engine. The Action Module manages these actions, while the Reward Module computes rewards based on the actions of each cluster and determines whether to initiate the learning process anew.

Container Orchestration, comprising the Resource Monitor, Task Scheduler, Cluster Management, and Low-latency Management modules, handles various tasks.

Resource Monitor tracks real-time resource consumption metrics for each active node in the clusters. When Task Scheduler seeks access to cluster information, Resource Monitor retrieves executable cluster node information from Cluster Management and provides it. Cluster Management oversees the addition or removal of cluster nodes.

Task Scheduler, responsible for containerizing N applications received from Distributed RL Engine and

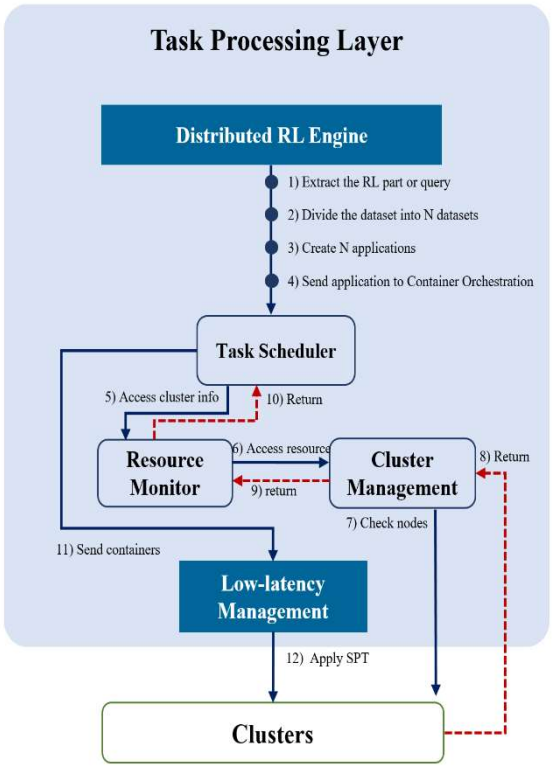


Figure 6: Task Processing Layer workflow

assigning containers to clusters, compares the cluster resource information obtained from Resource Monitor with the application's resource requirements.

It then maps them to maximize cluster resource utilization.

Upon transferring containers to clusters, they undergo Low-latency Management. This module minimizes the impact of network latency and employs Safe Proper Time (SPT) protocol during data transmission. SPT selects one of the batch, sequential, or integrated methods based on the data size [11].

Algorithm 1 and Algorithm 2 depict the execution algorithms of the Distributed RL Engine and Container Orchestration, respectively.

Algorithm 1: Distributed RL Engine execution algorithm

ALGORITHM: Distributed RL Engine

INPUT: request of process
OUTPUT: Apps

```
BEGIN
  # Extract the RL parts and load the data.
  RLpart := ExtractRL(Process);
  reqData := ChooseData(RLpart);
  dataSource := Distributed Data Storage
                <- RequestData(reqData);
  #N = number of datasets
  NumDataSet = DataSource / N;
  # Split the dataset.
  BEGIN
  for i in range(N + 1):
  if (i == 0):
      dataset[i] =DataSource[i : NumDataSet + 1];
  else:
      dataset[i] = DataSource[i * NumDataSet +
                   1 : (i+1) *NumDataSet +1];
  ENDFOR
  # Creating the application and transmitting it
  through Container Orchestration

  BEGIN
  for i in range(N + 1):
  app[i] := WrapData(dataset[i]);
  Container Orchestration < - TransferApps(App);
  ENDFOR

END
```

Algorithm 2: Container Orchestration execution algorithm

ALGORITHM: Container Orchestration

INPUT: Apps
OUTPUT: Containers

```
BEGIN
    containers := ContainerizeApps(Apps);
    Resource Monitor <- AccessInfo();
    Cluster Management <- AskResource();
    clusterInfo := Clusters <- CheckNodes();
    Low_latency_Management <-
                DeployContainers(containers
                );
    Clusters <- ApplySPT(containers);
END
```

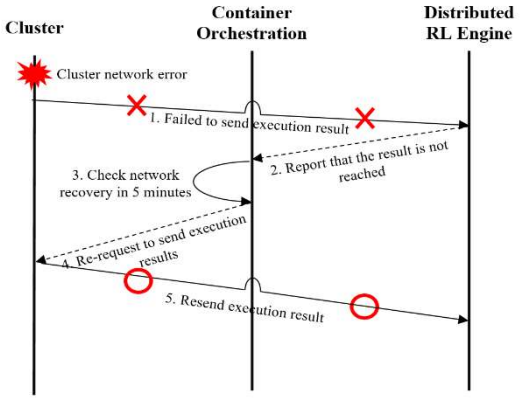
3.4 Network latency and error handling methods

Cases where network latency and failures may occur in the proposed system are as follows:

- Case 1: In the event of a network error within the cluster, where the application execution results fail to reach the Distributed RL Engine successfully.
- Case 2: If a cluster node encounters an error during application execution, rendering normal application execution impossible.

For Case 1, let's assume that the cluster has completed the application execution, but due to temporary network instability, the results are not being returned. The fault handling mechanism for Case 1 is divided into two approaches depending on whether network recovery is possible.

If the cluster fails to transmit the execution results to Distributed RL Engine, Distributed RL Engine reports the issue to Container Orchestration. Subsequently, Container Orchestration waits for network connectivity recovery for 5 minutes. If the network is restored, as depicted in Figure 7, Container Orchestration requests the cluster to resend the execution results. If the network remains



unrecovered, as shown in Figure 8, Container Orchestration disconnects from the existing cluster and creates a new one. Then, it redeploys the containerized application and handles the execution within the new cluster, ensuring the results are

Journal of Theoretical and Applied Information Technology
15th May 2024. Vol.102. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

successfully transmitted afterward.

Figure 7: Handling failure in Case 1: network recovery

Figure 8: Handling failure in Case 1: network unrecoverable

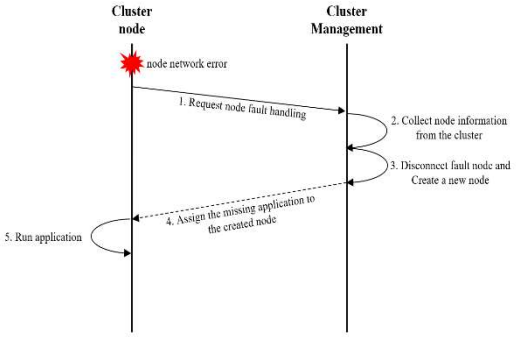


Figure 9: Handling failure in Case 2

When error resolution is requested for a cluster due to network errors on a cluster node in Case 2, Cluster Management module of Container Orchestration receives information about the faulty cluster node. It then proceeds to disconnect the network connection of the malfunctioning node within the cluster and initiates the creation of a new node. Subsequently, the malfunctioning application is assigned to the newly created node for execution. Users can monitor this error resolution process in real-time through User Interface.

3.5 Execution process

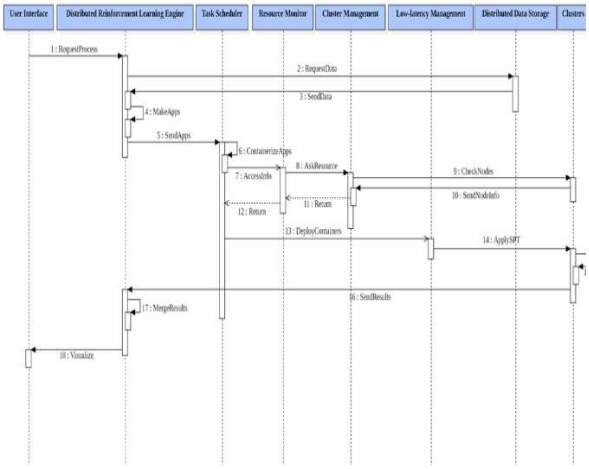
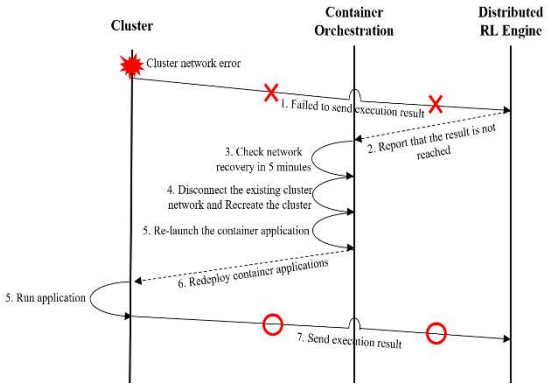


Figure 10: Sequence diagram of the proposed system

Figure 10 shows the overall flow of the proposed system as a sequence diagram.

- RequestProcess() : Request process execution
- RequestData() : Request data for RL analysis
- SendData() : Send the requested dataset and RL model data
- MakeApps() : Create applications according to the



number of divided datasets

- SendApps() : Send N applications to Container Orchestration
- ContainerizeApps() : Containerizes applications
- AccessInfo() : Access the cluster resource information
- AskResource() : Request resource information
- CheckNodes() : Checks viable cluster nodes
- SendNodeInfo() : Send node resource information
- DeployContainers() : Deploy containers
- ApplySPT() : Apply the SPT protocol
- RunApps() : Run the applications
- SendResults() : Send execution results
- MergeResults() : Merge N results in one result
- Visualize() : Visualize application execution results

4. EXPERIMENTS AND DISCUSSION

4.1 Experiments and results

In this study, we conducted two experiments to evaluate the performance of the proposed system. Firstly, we measured the response failure rate based on the number of cluster nodes. The response failure rate was defined as cases where the Distributed RL Engine did not receive the execution results properly after the applications were executed in the clusters. Five different clusters were configured, and the response failure rate was recorded for each cluster node count. Through this initial experiment, we aimed to determine the optimal number of cluster nodes that would maximize system performance.

The second experiment involved comparing the processing time between the proposed system and a centralized processing method. Building upon the results of the first experiment, where we identified the optimal number of nodes, we proceeded to

measure and compare the processing time of the proposed system with that of a centralized processing approach, where all data processing occurs on a single server.

The experimental setup for both experiments is detailed in Table 1 below.

Table 1: Experiment environment and system specifications

Distributed RL Engine Server	OS	Ubuntu 22.04
	CPU	Intel(R) Core(TM) i7-6850K CPU @ 3.60 GHz
	GPU	GeForce STX 1080Ti x 4
	RAM	64 GB
	Storage	SSD 256 GB HDD 2 TB
Container Orchestration		Kubernetes, Docker
Database		MongoDB Community 6.0
Node Configuration	OS	VMware virtual machine, Ubuntu 22.04
	RAM	16 GB
	Storage	32 GB

In the first experiment, following the execution of applications in the clusters, the resulting execution data was transmitted to Distributed RL Engine in h5 format. Table 2 presents the response failure rates observed with an incremental increase in the number of nodes per cluster, ranging from 3 to 10. The response failure rate represents the average of measurements obtained from 40 experiments, with 10 trials conducted for each node count. As depicted in Table 2, the response failure rate demonstrates an upward trend with the increase in the number of nodes. This observation is anticipated, as the transmission of execution results from the clusters to Distributed RL Engine is influenced by network conditions, necessitating time for the determination and adjustment of cluster resource information.

Table 2: Response failure rate (%) based on the cluster node count

		Cluster name				
		C1	C2	C3	C4	C5
Node Count	3	0.64	0.32	0.85	0.54	0.77
	5	1.54	1.28	2.02	1.33	1.89
	7	3.62	3.23	4.08	3.86	3.99
	10	6.54	5.82	6.31	6.42	6.75

The second experiment entails a comparison of processing speeds between the proposed system and a centralized processing approach. Building upon the findings of the first experiment, which indicated that clusters comprising three nodes exhibited the lowest response failure rate, we formed three clusters, each consisting of three nodes. Table 3 presents the execution times for each cluster as well as the centralized processing method. The execution time is defined as the duration from the initiation of application execution in the clusters to the completion of data return Distributed RL Engine.

Table 3: Performance comparison between the proposed system and the centralized processing approach

	Cluster name	Execution time	Average execution time
Proposed system	Cluster 1	1,012.98s	1,220.70s
	Cluster 2	645.02s	
	Cluster 3	2,004.12s	
Centralized processing method	-	-	37,691s

In Table 3, the average execution time of the proposed system represents the mean of the execution durations for Clusters 1, 2, and 3. The average execution time in the proposed system was approximately 1,220 seconds, whereas the centralized processing method required 37,691 seconds, equivalent to approximately 10 hours and 40 minutes. This is because during the learning process in the centralized processing approach, significant CPU time is consumed by other processes aside from the learning process itself. In fact, as the number of epochs increases in the centralized processing approach, the memory occupancy of the learning process rises, resulting in longer execution times for the learning process. Additionally, scheduling issues exacerbate the situation, leading to a substantial increase in the execution time of the learning process. Therefore, it is evident that the proposed system exhibits significantly faster processing capabilities compared to the centralized processing method.

4.2 Comparison analysis

The system proposed in this paper enables data analysis even with limited resources by containerizing and performing cluster distributed analysis of data analysis applications. Since the proposed system utilizes low-spec computing resources as cluster nodes, it may be somewhat challenging to run services other than container applications. Therefore, the proposed system is configured to exclusively execute container applications. In this section, we compared existing ML systems such as RapidMiner [14], Dataiku [15], and KRAKEN based on criteria including workload support, interface convenience, infrastructure environment, application deployment unit, ML support, and initial installation cost [13].

In terms of workload support, both RapidMiner and Dataiku support workload automation, whereas KRAKEN does not. The proposed system partially supports workloads. All four systems have GUI-based interfaces, ensuring user accessibility. While RapidMiner and Dataiku can be deployed in cloud, fog, and edge infrastructure environments, KRAKEN is only feasible in a cloud environment. The proposed system is initially deployed in a cloud environment but can be expanded to fog or edge environments. RapidMiner allows application deployment via VMs and containers, whereas Dataiku and KRAKEN only deploy applications via VMs, potentially leading to less resource efficiency compared to container deployment. The proposed system is deployed via containers, allowing efficient resource utilization similar to RapidMiner, enabling collaboration among various companies.

The ML functionalities supported by each system are summarized in Table 4. Although RapidMiner supports a wide range of ML functionalities, its initial setup is challenging, and it is not widely adopted by companies in South Korea. Similarly, Dataiku incurs significant initial installation costs, making it less suitable for small-scale companies. However, the proposed system is easy to set up and has low initial installation costs, making it suitable for adoption by small-scale companies. Table 4 summarizes the characteristics and differences among the four systems.

Table 4: Comparison of features between the existing ML system and the proposed system

	RapidMin-er [14]	Dataiku [15]	Kraken	proposed system
Workload Support	Automat-able	Automat-able	Not Automat-able	Partial Workload Support
Interface Convenienc-e	Easily Accessible through GUI	Easily Accessible through GUI	Intuitive Interface Design	Intuitive GUI-based Interface
Infrastructure Environment	Cloud, Fog, Edge	Cloud, Fog, Edge	Cloud	Expandable to Fog, Edge
Application Deployment Unit	VM, Container	VM	VM	Container
ML Support	Model Prediction, Resource Provisioni-ng, ML Orchestrat-ion	Model-specific Training, Data Pipeline	Simultan-eous Exploration of ML Algorith-ms	Distributed Reinforcem-ent Learning
Initial Installation Cost (small-scale businesses)	Difficulty in Initial Setup	High Initial Installation Costs	Easy Initial Setup	Easy Initial Setup

5. CONCLUSION

This paper proposes a container orchestration system for distributed reinforcement learning to facilitate efficient reinforcement learning data analysis applications. Among the components of the proposed system, Distributed RL Engine serves as an engine optimized for distributed reinforcement learning, allowing for dataset partitioning and application automation without the need for users to directly consider the distributed processing, thus offering advantages. Container Orchestration sets idle computing resources as cluster nodes, enabling the utilization of low-spec computing resources, leading to cost savings and increased resource utilization and efficiency. Additionally, the Low-latency Management module of Container Orchestration adapts transmission methods based on the size of the data to mitigate network latency, enabling relatively fast data transmission.

Through experimentation and comparative analysis, the performance of the proposed system was evaluated. The results indicate that the proposed system offers advantages in terms of ease of system setup compared to existing ML systems, particularly in terms of initial installation costs. Moreover, it demonstrates superior performance improvements in data analysis time differences compared to centralized processing methods. Thus, the container orchestration system for distributed reinforcement learning proposed in this paper can be considered a suitable solution for business process analysis in small-scale enterprises.

Although this paper aimed to build a system optimized for reinforcement learning by incorporating the Distributed RL Engine, it acknowledges limitations regarding the implementation of analysis extraction functionalities within the Distributed RL Engine. Furthermore, as containers pass through the Low-latency Management module upon deployment in the cluster, the SPT method of this module presents security concerns related to data encryption, necessitating research to address network security issues.

REFERENCES

[1] C.W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, “Big data analytics: a survey,” Journal of Big Data, vol. 2, no. 1. Springer Science and Business Media LLC, Oct. 01, 2015, doi: https://doi.org/10.1186/s40537-015-0030-3.

[2] G. Rjoub, J. Bentahar, O. Abdel Wahab, and A. Saleh Bataineh, “Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems,” Concurrency and Computation: Practice and Experience, vol. 33, no. 23. Wiley, Jul. 27, 2020, doi: https://doi.org/10.1002/cpe.5919.

[3] S. Y. Jang, H. J. Yoon, N. S. Park, J. K. Yun, and Y. S. Son, “Research Trends on Deep Reinforcement Learning,” Electronics and Telecommunications Trends, vol. 34, no. 4, pp. 1–14, Aug. 2019, doi: https://doi.org/10.22648/ETRI.2019.J.340401

[4] S. Gronauer and K. Diepold, “Multi-agent deep reinforcement learning: a survey,” Artificial Intelligence Review, vol. 55, no. 2. Springer Science and Business Media LLC, pp. 895–943, Apr. 15, 2021, doi: https://doi.org/10.1007/s10462-021-09996-w.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” IEEE Signal Processing Magazine, vol. 34, no. 6. Institute of Electrical and Electronics Engineers (IEEE), pp. 26–38, Nov. 2017, doi: https://doi.org/10.1109/msp.2017.2743240.

[6] S.-Y. Gu, S.-J. Moon, and B.-J. Park, “Reinforcement learning multi-agent using unsupervised learning in a distributed cloud environment,” International Journal of Internet, Broadcasting and Communication, vol. 14, no. 2, pp. 192–198, May 2022, doi: https://doi.org/10.7236/IJIBC.2022.14.2.192.

[7] S.Y.Noh, Cloud computing for everyone, Jpub, 2022

[8] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, “Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions,” ACM Computing Surveys, vol. 54, no. 10s. Association for Computing Machinery (ACM), pp. 1–35, Jan. 31, 2022, doi: https://doi.org/10.1145/3510415.

[9] Docker, https://aws.amazon.com/ko/docker/

[10] Orchestration big data platform, http://www.riss.or.kr/

[11] S.Y.Gu, S.-J. Moon, and B.-J. Park, “Agent with Low-latency Overcoming Technique for

Distributed Cluster-based Machine Learning," International Journal of Internet, Broadcasting and Communication, vol. 15,no. 1, pp. 157–163, Feb. 2023, doi: https://doi.org/10.7236/IJIBC.2023.15.1.157.

[12] B.H.Yoo, Devrani Devi, H.W.Kim, H.J.Song, G.M.Park, and S.W.Lee, "A Survey on Recent Advances in Multi-Agent Reinforcement Learning," Electronics and Telecommunications Trends, vol. 35, no. 6, pp. 137–149, Dec. 2020, doi:https://doi.org/10.22648/ETRI.2020.J.350614.

[13] M. Barika, S. Garg, A. Y. Zomaya, L. Wang, A. V. Moorsel, and R. Ranjan, "Orchestrating Big Data Analysis Workflows in the Cloud," ACM Computing Surveys, vol. 52, no. 5. Association for Computing Machinery (ACM), pp. 1–41, Sep. 13, 2019, doi: https://doi.org/10.1145/3332301.

[14] Hofmann, Markus, and Ralf Klinkenberg, eds. "RapidMiner: Data mining use cases and business analytics applications", CRC Press, 2016

[15] Liermann, Volker. "Overview machine learning and deep learning frameworks." The Digital Journey of Banking and Insurance, Volume III: Data Storage, Data Processing and Data Analysis, 2021, pp.187-224.